DOCUMENT RESUME

ED 388 274 IR 017 423

AUTHOR Matthews, James W.; And Others
TITLE VideoScheme: A Research, Authoring, and Teaching Tool for Multimedia.
PUB DATE 94
NOTE 7p.; In: Educational Multimedia and Hypermedia, 1994. Proceedings of ED-MEDIA 94--World Conference on Educational Multimedia and Hypermedia (Vancouver, British Columbia, Canada, June 25-30, 1994); see IR 017 359.
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Authoring Aids (Programming); Computer Software Development; Educational Technology; *Interactive Video; Multimedia Materials; Programming Languages; *Research Tools
IDENTIFIERS Digital Data; *Multimedia Technology; Prototypes

ABSTRACT

The availability of digital multimedia technology poses new challenges to researchers, authors, and educators. even as it creates new opportunities for communication. VideoScheme, a prototype video programming environment is described, along with its applications in research, authoring and education. In terms of research, VideoScheme can help eliminate information redundancy by detecting "cuts" between segments; in addition, VideoScheme's monitor object enables automatic video analysis to be applied to interactive multimedia. As an authoring tool, VideoScheme makes it possible to write applications in minutes or hours because of its rapid-turnaround interpreted language. In terms of education, the programmable multimedia environment of VideoScheme makes it a useful tool for exposing computer science students to digital media in a way that is concrete but one that does not restrict experimentation. (Contains 12 references.) (AEF)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☐ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED 388 274

VideoScheme: A Research, Authoring, and Teaching Tool for Multimedia

J. MATTHEWS, F. MAKEDON, AND P. GLOOR
Department of Mathematics and Computer Science
Dartmouth College, Hanover, NH, 03755, USA
E-Mail: James.W.Matthews@dartmouth.edu

Abstract: The availability of digital multimedia technology poses new challenges to researchers, authors, and educators, even as it creates new opportunities for rich communication. This paper suggests interactive computer programming as a fruitful approach to these challenges. VideoScheme, a prototype video programming environment, is described along with promising applications.

Introduction

The advent of affordable digital audio and video has unleashed new possibilities for interactive multimedia.[3] But it has also raised a host of questions, of how best to acquire, analyze, edit, and deliver digital media products. To answer these questions researchers and practitioners need new tools, tools with enough flexibility to accommodate new demands, explore original hypotheses, and prototype novel applications. We believe that VideoScheme, a programming environment for digital media, offers a valuable example of such a tool.[4]

System Description

VideoScheme is implemented as an application for the Apple Macintosh, written in C and totaling approximately 100KB of executable code. It provides a visual browser for viewing and listening to digital movies, using Apple's QuickTime system software for movie storage and decompression.[9] The browser displays video and audio tracks in a time-line fashion, at various levels of temporal detail. Clicking on the video tracks displays individual frames in a "flip book" fashion, while clicking on the audio track plays it back; clicking twice in rapid succession plays back both audio and video.

As a visual interface to digital media VideoScheme is nothing out of the ordinary; what separates it from conventional computer-based video editors is its programming environment. VideoScheme includes an interpreter for the LISP-dialect Scheme, built on the SIOD (Scheme-in-one-Defun) implementation, along with text windows for editing and executing Scheme functions.[10] Functions typed into the text windows can be immediately selected and evaluated. The environment, while deficient in debugging facilities, offers such standard LISP/Scheme programming features as garbage collection and a context-sensitive editor (for parentheses matching). In addition it offers a full complement of arithmetic functions for dynamically-sized arrays, an important feature for handling digital video and audio.

Scheme was chosen over other alternatives (such as Tcl, Pascal, and HyperTalk) for a number of reasons. Scheme treats functions as first class objects, so they can be passed as arguments to other functions. This makes it easier to compose new functions out of existing ones, and adds greatly to the expressive power of the language. Scheme is also easily interpreted, a benefit for rapid prototyping. Scheme includes vector data types, which map very naturally to the basic data types of digital multimedia, namely pixel maps and audio samples. Finally, Scheme is easily implemented in a small amount of portable code, an advantage for research use. The most significant drawback to using Scheme is the programming syntax, which non-programmers (and even some programmers) find difficult to use. If we were to start again we might consider Logo or Dylan, languages with the positive properties of Scheme but with more attractive syntax.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Gary H. Marks
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

IR017493
BEST COPY AVAILABLE
2

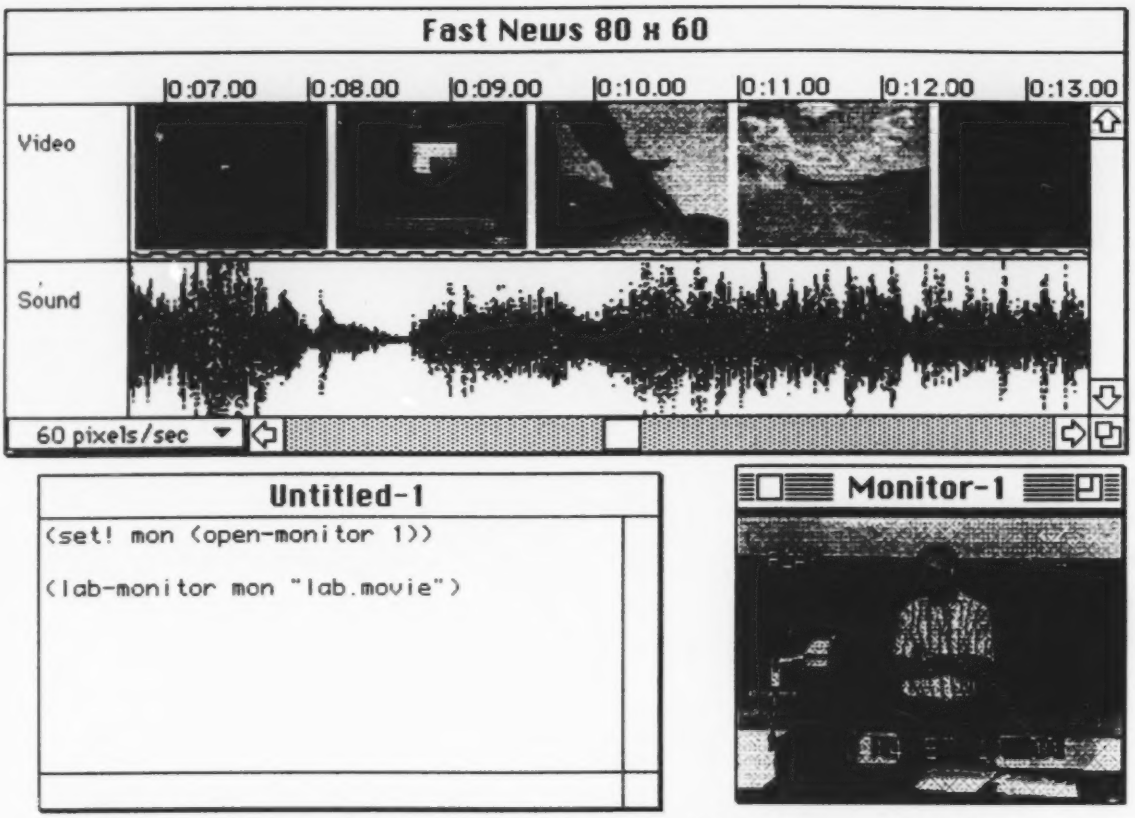

Figure 1 VideoScheme user interface, with movie (top), monitor (right), and program editor (left).

Language Description

VideoScheme extends the Scheme language to accommodate digital media. In addition to the standard number, string, list, and array data types VideoScheme supports the following objects:

movie	-- a stored digital movie, with one or more tracks.
track	-- a time-ordered sequence of digital audio, video, or other media.
monitor	-- a digital video source, such as a camera, TV tuner, or videotape player.
image	-- an array of pixel values, either 24-bit RGB or 8-bit gray level values.
sample	-- an array of 8-bit Pulse Code Modulation audio data.

These objects are manipulated by new built-in functions. Movies can be created, opened, edited, and recorded:

```
(new-movie)                    ; creates and returns a reference to a new movie

(open-movie filename)          ; opens a stored movie file

(cut-movie-clip movie time duration)
                               ; moves a movie segment to the system clipboard
(copy-movie-clip movie time duration)
                               ; copies a movie segment to the system clipboard
(past-movie-clip movie time duration)
                               ; replaces a movie segment with the segment on the clipboard
(delete-track movie trackno)
                               ; removes a movie track
```


BEST COPY AVAILABLE

```
(copy-track movie trackno target)
                                ; copies a movie track to another movie
(record-segment monitor filename duration)
                                ; records a segment of live video from the monitor to a file
```

Images and sound samples can be extracted from movie tracks or monitors, and manipulated with standard array functions:

```
(get-video-frame movie trackno time image)
                                ; extracts a frame from a video track
(get-monitor-image monitor image)
                                ; copies the current frame from a video source
(get-audio-samples movie trackno time duration samples)
                                ; extracts sound samples from an audio track
```

With this small set of primitive objects, and small number of built-in functions, we can rapidly build a wide variety of useful functions with applications in research, authoring and education.

Research Applications

With powerful computers and digital media it is possible to build systems that perform automatic analysis of video and audio data. The results of this analysis can be used to make existing applications (such as indexing, retrieval, and editing) more efficient, or to enable entirely new applications.[6, 11, 12] It would certainly be desirable if computers could perform these analytical tasks as well as a trained human; but short of such a breakthrough in artificial intelligence there are numerous less-ambitious goals to be pursued by researchers in this field.

One important goal is an automatic system for dividing digital video into segments, by detecting the "cuts" that divide one shot from another. There is no perfect definition of a "cut," but generally the term refers to a sharp discontinuity in a video stream, such as the break between two recording bursts in unprocessed video, or the point where two clips were concatenated in the editing process. Once cuts are found the segments they define can be represented by a subset of the segment's data (e.g. the first and last frames), since the continuity of the segment ensures a great deal of information redundancy. This reduction of a potentially long segment to a few frames is a significant boon to a number of applications, such as indexing, logging, navigation, and editing, since those tasks may be performed on a greatly reduced set of data.

A number of algorithms have been proposed for automatic cut detection, and one of the advantages of VideoScheme is that such algorithms can be implemented as compactly as their mathematical formulation. For example, a simple measure of visual continuity is a sum of pointwise differences in gray value or color. Such a test can be performed by the following fragment of VideoScheme code:

```
(adiff frame1 frame2 delta)     ; subtract arrays of gray value
(aabs delta)                    ; compute absolute difference values
(atotal delta)                  ; sum differences
```

Scene changes trigger large pointwise differences, but this measure is also very sensitive to camera motion and zooming, which may change every pixel without introducing a new scene. Refinements of this algorithm, such as one that counts the differences that exceed a threshold, have a similar weakness. So there appears to be some value in a test that is not so spatially sensitive, such as the difference between summed gray values:

```
(- (atotal frame1) (atotal frame2)) ; subtract summed gray values
```

This measure is insensitive to camera pans and zooms, but it is also insensitive to actual cuts, since the average gray level may not change dramatically across the cut. A more reliable indicator is the gray level or color histogram. Using VideoScheme's built in color histogram function we can easily compute this measure:

```
(get-color-histogram64 frame1 histogram1) ; compute 64-bucket color histograms
(get-color-histogram64 frame2 histogram2)
(adiff histogram1 histogram2 delta)       ; subtract histograms
(aabs delta)                              ; compute absolute differences
(atotal delta)                            ; sum differences
```

This test can be further refined, by breaking each frame into a number of sub-frames and discarding the ones with above-median changes, or counting the number of changes that exceed a threshold; either modification is quickly implemented in VideoScheme, and each makes the algorithm more robust against local phenomena such as object motion.

While histogram comparison is widely considered a robust solution to detection of simple camera breaks, the general problem remains a fertile area for new approaches. In recent months novel algorithms have been proposed for detecting gradual transitions, and for using motion-sensitive measures in conjunction with a projection detecting filter.[12, 7] The future is bound to bring still more proposed algorithms, targeted at specific video sources and applications. We believe that VideoScheme's flexibility makes it a useful vehicle for this research. VideoScheme's high-level primitives and rapid-turnaround programming environment make programs compact and prototyping very rapid. Nagasaka and Tanaka's histogram-based algorithm has been implemented as 25 lines of VideoScheme, and the Otsuji-Tonomura filter was implemented in a couple of hours.[6, 7] We aim to implement and evaluate new cut detection algorithms as they are proposed, and hopefully develop our own improvements.

Another area of research is the application of automatic video analysis to interactive multimedia. Linblad has termed this "computer-participative multimedia" since the computer is actively involved, reacting to the content of the video data.[2] VideoScheme's monitor object makes it possible to easily implement such applications. For example, this VideoScheme code fragment implements the core of a video room monitor:

```
(set! camera (open-monitor 1))                ; create a monitor object for a camera
(get-monitor-image camera baseline)           ; capture a baseline image
(while t
 (get-monitor-image camera new-image)                ; capture a new image
  (if (image-diff baseline new-image)                ; compare to baseline
   (record-segment camera "monitor-movie" 5.0)))     ; record 5 seconds of video
```

With a camera pointed at a door this fragment keeps a running log of all the people who enter and leave the room. With the addition of a sound playback command this function could also serve as a video answering machine: people approaching a closed door could be automatically prompted to leave a videotaped message. This fragment assumes that any significant change in the image striking the camera represents a person. More sophisticated analysis functions could look for more specific phenomena. So, for example, they could automatically record television programs with a certain opening screen or on-screen logo.

Authoring with Programming

Digital media technology has enabled more than the creation of movies on computers; it has also made possible new sorts of information systems. One example is the *Parallel Computation*, a CD-ROM that integrates digital video, audio, animation, and hypertext to communicate the proceedings of an academic conference.[5] The unusual nature of the system posed numerous challenges to the development staff, challenges that were only partly addressed by existing authoring tools. For example, the centerpiece of the proceedings is 8 forty-five minute synchronized audio/video presentation of speech with overhead slides. It was sometimes necessary to move audio tracks independently of the accompanying video, to accommodate sound editing software. Our commercial tools did not provide a way to do this without re-compressing six hours of animations, but VideoScheme was quickly adapted to the task. Earlier in the process we needed to remove silences from the original 12 hours of sound tracks, a painstaking manual process. Had VideoScheme been available at that time it would have been a simple function:

```
(while (< time (get-movie-duration movie)); loop through whole movie
 (if (silence? movie trackno time 0.1)     ; if there is 0.1 seconds of silence
  (cut-movie-clip movie time 0.1)          ; then remove that segment
  (set! time (+ time 0.1))))               ; otherwise move on to next segment
```

We also removed noise words (such as "um" and "ahh") by hand, and we believe that VideoScheme could be extended to assist in this step as well.

It was clear from our experience that while existing tools offer high-quality solutions for problems in their domain, they are often poorly suited to the ad hoc tasks that arise in innovative multimedia authoring. In such cases a programmable system may provide the necessary flexibility to turn a tedious, manual process into an automatic operation. It may also permit entirely novel operations. In his work on a programmable graphics editor Eisenberg noted that such an editor can produce effects, such as fractals and recursive Escher-like designs, that would be near-impossible with manual tools.[1] Likewise we can imagine VideoScheme programs to implement algorithmically defined effects, such as fades and wipes, that could hardly be achieved any other way.

A critical concern with authoring tools is the speed with which new applications can be developed. Traditional programming environments (featuring compiled languages such as C and C++) offer complete access to the computer hardware and data, but can require months or years of effort to produce a usable tool. Object-oriented frameworks accelerate the process somewhat, but even then many details must be coded by hand. By contrast, a rapid-turnaround interpreted language like VideoScheme makes it possible to write applications in minutes or hours. Tool authors do not need to master the low-level details, and powerful operations can be composed out of very high-level primitives and previously-written functions.

Education in Multimedia

Education is an obvious application for multimedia, but it is also true that students will need education **in** multimedia. As more data becomes available in multimedia form it will become more important for students to understand the fundamentals of the technology, in order to use this form of communication and to be discerning consumers of multimedia information. Commercial multimedia tool developers are one source of teaching tools, since their products are typically very polished and focused on the task of making multimedia production as easy as possible.[8] But there is also a call for more basic, flexible tools, that expose more of the underlying technology and permit easy experimentation with features that might not be included in a commercial package.

We believe that VideoScheme is such a tool, and to that end made it the subject of a project in a multimedia seminar for undergraduates and graduate students in computer science. All the students had programming experience, but many had not used Scheme or a LISP dialect and none had written programs to manipulate digital video. The lack of debugging facilities in VideoScheme was a troublesome obstacle, as was the parentheses-heavy syntax of Scheme. Nonetheless, after a 90-minute tutorial most students were able to write functioning VideoScheme programs to perform a specified editing task and implement a crude cut detection algorithm. They were also able to informally analyze the performance of the cut-detection algorithm (which was based on an average gray-level test) and thereby make classroom discussion of its faults more informed and concrete. The flexibility of the system also prompted the students to suggest improvements and new applications, including the video answering machine mentioned above.

While learning to program is a significant obstacle for most students, a programmable multimedia environment appears to be a useful tool for exposing computer science students to digital media in a way that is concrete without unnecessarily restricting their experimentation. By creating multimedia programs students can gain appreciation for the ones they merely use, in the same way that writing fosters appreciation for literature. In the best of cases such an experience can help students to imagine and implement completely original applications without the expertise required by conventional low-level programming tools.

Conclusion

Digital media are producing exciting, and rapid changes in the way we communicate. There is much research to be done, and much that the multimedia producers and consumers of the future should learn, in order to fully exploit the potential of this technology. We believe that programmable environments such as VideoScheme

provide the flexibility to meet this dynamic challenge, and we look forward to finding new applications for this system.

[1] Eisenberg, M. "Programmable Applications: Interpreter Meets Interface." MIT Artificial Intelligence Memo 1325, October 1991.

[2] Linblad, C. "The VuSystem: A Computer-Participative Multimedia Toolkit." MIT Laboratory for Computer Science, 1993.

[2] Linblad, C., Wetherall, D., and Tennenhouse, D. "The VuSystem: A Programming System for Visual Processing of Digital Video." TR-590, MIT Laboratory for Computer Science, 1993.

[3] Mackay, W. and Davenport, G. "Virtual Video Editing in Interactive Multimedia Applications." Communications of the ACM, 32:7, July 1989.

[4] Matthews, J., Gloor, P. and Makedon, F. "VideoScheme: A Programmable Video Editing System for Automation and Media Recognition." ACM Multimedia '93, Anaheim, CA, 1993.

[5] Metaxas, T., Cheyney, M., Gloor, P., Johnson, D., Makedon, F., and Matthews, J. "Conference on a Disk: An Experiment in Hypermedia Publishing." ED-MEDIA '94, Vancouver, Canada, 1994.

[6] Nagasaka, A. and Tanaka, Y. "Automatic Video Indexing and Full-Video Search for Object Appearances." IFIP Transactions A (Computer Science and Technology), vol. A-7, 1992.

[7] Otsuji, K. and Tonomura, Y. "Projection Detecting Filter for Video Cut Detection." ACM Multimedia '93, Anaheim, CA, 1993.

[8] Premiere. Adobe Systems Incorporated. Mountain View, CA.

[9] QuickTime. Apple Computer, Inc. Cupertino, CA.

[10] SIOD (Scheme-in-one-Defun). Paradigm Associates, Inc. Cambridge, MA.

[11] Ueda, H., Miyatake, T., and Yoshizawa, S. "IMPACT: An Interactive Natural-Motion-Picture Dedicated Multimedia Authoring System." CHI '91 Conference Proceedings, 1991.

[12] Zhang, H., Kankanhalli, A. and Smoliar, S. "Automatic partitioning of full-motion video." Multimedia Systems, 1:1, August 1993.